An Redundant Networking Channel to Support Reliable Communications in the Internet of Things Applications

Michael Ortiz[1], Yu Sun[1(✉)], Gilbert S. Young[1], and Qingquan Sun[2]

[1] Computer Science Department, California State Polytechnic University, Pomona, Pomona, USA
{mdortiz,yusun}@cpp.edu
[2] School of Computer Science and Computer Engineering, California State University, San Bernardino, USA
qsun@csusb.edu

Abstract. Within the context of the Internet-of-Things (IoT), the number of interconnected devices is increasing dramatically and allowing for access to physical data that was previously unimaginable. Physical data is rapidly changing which makes it important to keep networking connections active. Any drop in communication can lead to the loss of sensitive data. A redundant network connection is an attempt to utilize common networking solutions in order to decrease the likelihood of network downtime. It does this by adding a new level of abstraction to networking, allowing data to be sent over multiple networking solutions as if it were a single network, as well as an intelligent decision engine to determine the most optimized and reliable connection to use dynamically.

Keywords: IoT · Communication channel · Code generation

1 Introduction

The Internet-of-Things (IoT) is a rapidly growing area of technology that is impacting aspects of everyday life in both the working and domestic sectors. The basic idea of this concept is the pervasive presence around us of a variety of things or objects – such as Radio-Frequency Identification (RFID) tags, sensors, actuators, mobile phones, which through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals [1]. IoT is gaining attention because of its flexibility and cost-effective nature, allowing access to data that was previously unimaginable. From a statistical standpoint, IoT devices have overtaken the human population by reaching 11 billion devices in 2011 and this number is expected to reach 24 billion devices by 2020 [2].

IoT is powerful because of its flexibility and ability to connect to multiple devices. With connections to different components, there comes the need for secure and reliable communication channels. Digitization of physical objects means that those objects must perform the same operations without any extra

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2017
X.-L. Huang (Ed.): MLICOM 2016, LNICST 183, pp. 283–292, 2017.
DOI: 10.1007/978-3-319-52730-7_29

complexity. This must occur while also keeping data safe as it travels through networks. Current communication systems such as Wi-Fi and Bluetooth have had years of work put into them in order to allow secure transfer of data and is active in numerous amounts of current mobile devices. Other communications methods such as RFID are not as prevalent and would not be able to connect to multiple devices.

Open Problem: Current networking solutions are not fit for the demand of consistent data transmission from IoT devices as high-level languages abstract I/O communications and expect network failures. Networking applications today can function with an interrupted connection for short periods of time to counter unreliable connections. However, if the IoT applications are programmed in the same fashion, it will inevitably undermine the full potential of IoT devices.

Interactions with the physical world are constant, thus, information can be gained or lost with any disconnection. General purpose networking techniques, such as TCP/IP, focus on reliably delivering packets rather than timing. More specialized networking techniques must come into development to use in IoT devices. Moreover, network time synchronization technology must be improved considerably. Networks must offer the possibility of timing coherency across multiple, distributed computations. Networking innovations will dramatically change the way distributed real-time software is designed.

Solution Approach $\Rightarrow$ Model-based network management and the abstraction of communication interfaces. To address these challenges, this paper presents a network model that combines common communication protocols together to provide reliable connections across multiple devices. A model-based framework for communication between the IoT devices and the server. All IoT systems follow the same concept of sending data from devices to a server which processes the data and sends to other clients. Each device may differ based on its functionality, but the nature of communication using the Internet applies to all the IoT systems. The approach presented in this paper abstracts the common elements and entities used in the implementation of communication channels in order to both simplify communication to the necessary messages and manage the network to reduce possible downtime.

The remainder of this paper is organized as follows. Section 2, we provide a motivating example for this paper. We list down the challenges faced while developing reliable communication system in Sect. 3 and present our solution to address these challenges in Sect. 4. We analyze the related work in Sect. 5 and conclude this paper as well as provide some insight on future work and scope of this framework, in Sect. 6.

2 Motivating Example

IoT brings the possibility of new devices to consumers and health systems are taking notice. Major academic research has gone into innovative solutions for mobile healthcare delivery and sensors. In particular, major advances were introduced in the mobile broadband and wireless internet m-health systems [4]. This widespread and unprecedented evolution of m-health systems and services in recent years has been reflected in a 2010 study by McKinsey estimated that the opportunities in the global mobile healthcare market are worth between 50 billion and 60 billion [5]. As healthcare data becomes increasingly profitable, so will the wireless technologies bundled with the devices. There will be multiple reasons to address the communications streams for health devices:

1. The ability to relay information constantly. It is expected that mobile healthcare devices continue to monitor the user at all times. Devices such as the Fitbit [6], Jawbone [7], Misfit [8], etc. need to monitor both active lifestyle activity as well as sleeping patters. These are devices that will consist of as little down time as possible.
2. Choosing the right mobile technology for price and communication. IoT devices must stay on for long periods of time. Some might be connected directly to an outlet, others will use batteries and some might even utilize passive radio transmission (RFID). The goal for any of these devices is to relay information to both the user and the management system for the data.
3. Dealing with communication failure. In the event that a device goes offline, it is imperative that the device reconnect in the simplest way possible for the user. Down time will reduce the usability of healthcare devices and can cause profit loss for businesses.
4. Security of wireless technologies. A common problem for modern devices is adapting to modern technologies while also keeping data safe. Sensitive health data need to be kept secure wherever the user might go which means that the mobile communication must be able to connect to all other IoT devices securely.
5. Standardization to allow adaptable communication. A standard protocol for sending and receiving health data will allow devices to adapt to the rapid advancement of technology. If a new devices comes along with better or changed hardware, standardizing the exchange of data will allow older devices to keep in communication.

With advancements in these areas, mobile health will be able to create a more focused effort at improving livelihood. By allowing devices constant information to a user's heartrate, food consumption, exercise routine, medical regiment, and more, we will be able to decrease the duration of hospital stays along with improve a doctor's knowledge of his or her patients. This comes too with a need to improve security. The amount of data that will be available needs to be kept safe so users will not be used maliciously. Trust that mobile health devices keep a user's information private will be just as important as the direct service. Users will feel more comfortable using communication devices that they are familiar with such as Wi-Fi or Bluetooth or adapt to newer technologies like RFID as long as they do not complicate the application. By keeping a developmental standard in current and future healthcare devices, businesses will be able to gain quicker market adaptation and access to other medical information.

3 Challenges

Abstraction of any computer system is never without complication which goes double for areas that require synchronization such as networks. Current IoT connection problems are as follows:

1. *The inconvenience of networking channel setup in IoT development.* Research development for IoT devices becomes a challenge as most devices do not keep common developer interfaces while active. Communication settings such as Wi-Fi and Bluetooth are hosted in locations that are not convenient to edit initially (e.g., using the Raspberry Pi development board, you need to connect to a monitor, use a keyboard and mouse to edit and control Bluetooth devices).
2. *The challenge of implementing the communication channel.* For instance, Bluetooth has the advantage of easy setup for clients, but programming Bluetooth is challenging, particularly for different types of devices and protocols. Creating a client to find Bluetooth devices must be synchronous as to not interrupt other device communications.
3. *For mission critical IoT applications, there lacks a reliable communication channel to ensure data integrity.* Relying on a single communication channel and protocol is not reliable enough. Most communication systems are expected to fail [8]. This means that allowing only one pathway for communication is yet truly reliable.
4. *The lack of an intelligent decision engine on choosing the most optimized communication channel.* With multiple communication, understanding which portal to send data through is imperative but IoT devices do not have decent space for large scale, dynamic efficiency scaling.

For this project, the goal was to help pave the way to reduce some of these challenges as well as demonstrate the areas that still need work.

4 Solution

To reduce the challenge of network connectivity and problems with multiple connections on a single device, a generic framework has been developed - ReliableConnection, that allows multiple communication services to be unified to perform a singular function. As previously mentioned, the program works by the abstraction of network communication services, such as Bluetooth and Wi-Fi, and then managed autonomously in order to decrease the chance of downtime for multiple devices.

The two main components for the framework is the Network class and the Protocol class. The Network consists of a linked list of Protocols and manages which Protocol will send or receive data. The Protocol is an interface that abstracts TCP/IP and Bluetooth to a simpler functionality. This allows the network to observe the casted protocol rather than deal with specific complications in the communication services. To allow these classes to work with minimal

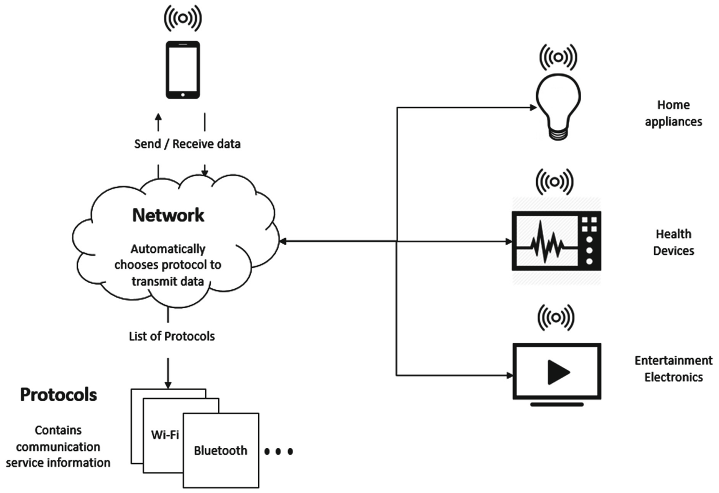

Send / Receive data
Network
Automatically chooses protocol to transmit data
List of Protocols
Protocols
Contains communication service information
Wi-Fi
Bluetooth
Home appliances
Health Devices
Entertainment Electronics

Fig. 1. Overview of ReliableConnection framework

supervision, an Observer pattern is implemented to both classes. The Network observes Protocols so that any changes to a Protocol will quickly notify the Network (Figs. 1, 2, 3 and 4. Addressed below are some of the ways the ReliableConnection framework addresses challenges mentioned in Sect. 3:

1. *Simplify the networking channel setup in IoT development with the default connection manager.* The connection manager is a built-in component in the development framework that handles the channel setup and initialization process. Allowing a framework to manage connection data will allow developers more time to focus on the logic of their code rather than reliability. Keeping a major factor in IoT development under stricter guidelines reduces the learning curve and invites increased innovation.
2. *Abstract the common communication channel implementations.* By creating a single interface for multiple communication channels, complexity of the communication is reduced while the benefits can be manipulated. For communication such as Bluetooth and TCP/IP, IoT devices can now switch between them without disruption. On the back-end, energy efficiency, distance from other devices and more can be monitored to allow the best communication possible for IoT devices.
3. *Enhance the data integrity by applying an intelligent and redundant connection channel.* Allowing ReliableConnection to manage networking activity provides standardization for network activity. By sending data over multiple communication services, critical IoT applications can increase reliability and chain devices with different hardware configurations.
4. *Support a optimized communication channel by runtime checking and machine learning techinques.* ReliableConnection's Network class provides developers a way to weigh different communication services against each other. By giving

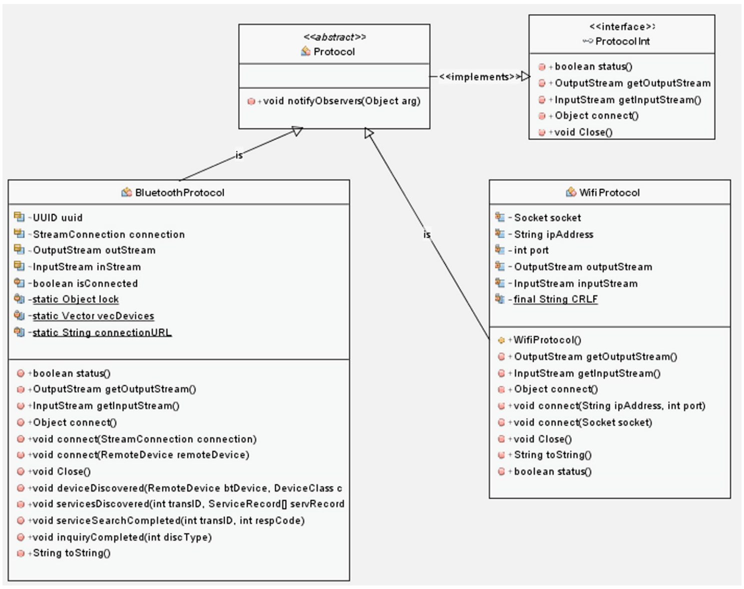
such measurements, data can pass through the different services depending on the developer's needs without complex code. In addition, the framework has a built-in verification engine to periodically check the performance using the two different connections, which enables a dynamic decision on which type of connection to use. We are also collecting the decision data, aligned with feature factors such as time, location, protocol, and data type, so that we will be able to apply machine learning to predict the best type of connection to use based on the actual application scenario.

With this framework, new communication services can be quickly added to the Network and contribute to the reliability without disrupting other Protocol behavior. A more descriptive discussion about the library follows.

4.1 Protocol

The Protocol class is a superclass that is modeled after the Observational design pattern and adaptable for future additions. As an Observable class, it contains a method to notify observers of any changes to the class, notifyObservers(). This method will pass along a reference of the Protocol to an Observer, the Network class.

The rest of the Protocol is defined in its interface implementation. The interface methods will be status(), getOutputStream(), getInputStream(), connect() and close(). The status() method is used to check if the Protocol is actively connected to a network. getOutputStream() has a return object of



<<abstract>>
Protocol
+void notifyObservers(Object arg)
<<implements>>
<<interface>>
Protocol Int
+boolean status()
+OutputStream getOutputStream
+InputStream getInputStream()
+Object connect()
+void Close()
is
is
BluetoothProtocol
-UUID uuid
-StreamConnection connection
-OutputStream outStream
-InputStream inStream
-boolean isConnected
-static Object lock
-static Vector vecDevices
-static String connectionURL
+boolean status()
+OutputStream getOutputStream()
+InputStream getInputStream()
+Object connect()
+void connect(StreamConnection connection)
+void connect(RemoteDevice remoteDevice)
+void Close()
+void deviceDiscovered(RemoteDevice btDevice, DeviceClass c
+void servicesDiscovered(int transID, ServiceRecord[] servRecord
+void serviceSearchCompleted(int transID, int respCode)
+void inquiryCompleted(int discType)
+String toString()
WifiProtocol
-Socket socket
-String ipAddress
-int port
-OutputStream outputStream
-InputStream inputStream
-final String CRLF
+WifiProtocol()
+OutputStream getOutputStream()
+InputStream getInputStream()
+Object connect()
+void connect(String ipAddress, int port)
+void connect(Socket socket)
+void Close()
+String toString()
+boolean status()

Fig. 2. UML diagram of the Protocol framework

java.io.OutputStream. for the communication standard contained in a Protocol object, Java's Output stream is a general way to send bytes of information across a network and will work with most networking devices. Similarly, getInputStream() is of java.io.InputStream. This allows the Protocol to have an input and output interface for the Network object to manipulate. The connect() method is used to implement any necessary function calls for connecting a network device and returns type object. The return for connect() is expected to be the stream for both getInputStream() and getOutputStream to utilize for that Protocol. Finally, close() is a standard method for closing the stream within the protocol for the Network or the user to utilize. The intention for the close() method is so that the Network object can close or reconnect to a Protocol automatically, for any debugging reasons.

4.2 Network

The Network is a class focused on the observation and management of Protocols for the client device. The way the Network class discovers and retrieves an active Protocol is through Java's Observer design pattern. Java's utility library has a simple Observer interface with an update method. This method, update(), is called when an Observable object's notifyObservers method is activated. The Observable method in this case is a Protocol. Once a Protocol is confirmed to be active, it will notify the network through notifyObservers() that it is a viable candidate for communication. The update method in Network then adds that Protocol to a Linked List.

When a user would like to send or receive data to another client they will go through the Network object. Network contains BufferedReader and PrintWriter objects for sending and receiving data. Network only allows users to use PrintWriter's println() and BufferedReader's readln() method. This abstraction from a stream formatter allows for dynamic allocation of IO streams. The Network class scans through Protocols and uses their IO streams

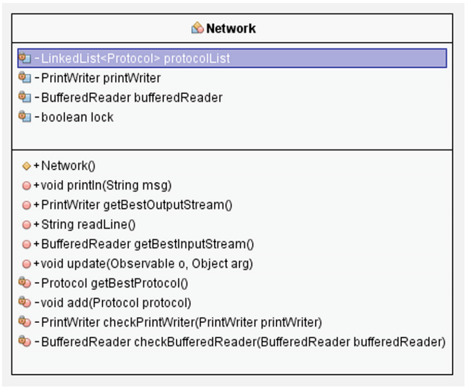

Network
-LinkedList<Protocol> protocolList
-PrintWriter printWriter
-BufferedReader bufferedReader
-boolean lock
+Network()
+void println(String msg)
+PrintWriter getBestOutputStream()
+String readLine()
+BufferedReader getBestInputStream()
+void update(Observable o, Object arg)
-Protocol getBestProtocol()
-void add(Protocol protocol)
-PrintWriter checkPrintWriter(PrintWriter printWriter)
-BufferedReader checkBufferedReader(BufferedReader bufferedReader)

Fig. 3. UML diagram of the Network framework

interchangeably within BufferedReader and PrintWriter. What the user will see is Network.println(String msg) and Network.readln() functions. The Network.println(String msg) function will send a string to all output streams available while the Network.readln() function will return a string from the best input stream available.

5 ReliableConnection in Action

To demonstrate the functionality of this framework, a multi-client chat application was created and hosted on Windows, Android and the Raspberry Pi microprocessor board. The client is given both the IP address of a server as well as discovers the Bluetooth hardware affiliated with the server. The Server will then relay messages from one client to any other client connected with handler threads. This application helps demonstrate how the ReliableConnection framework will make data transfer simpler for IoT devices. Each chat client can efficiently transfer data to a server using either TCP/IP or Bluetooth without the developer having to directly send data over each protocol. The clients will not have the challenge of complex networking design since the Network class in ReliableConnection will manage it. At the same time, the connection reliability for each client is increased since it can communicate over Bluetooth and TCP/IP interchangeably. Now that the framework manages each protocol in one area, information about each protocol can be compared to one another. With this data, intelligent IoT design is realized, dynamic communication is now simple and efficient. A more in-depth look of the client and the server is provided below.

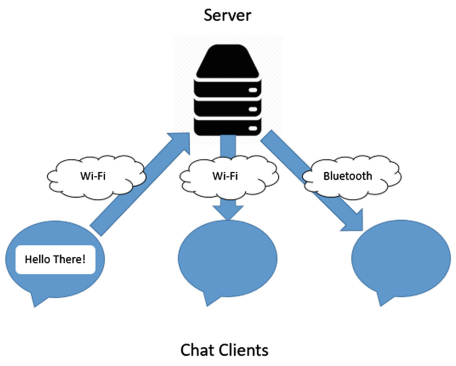


Fig. 4. The chat application. Clients send and receive data while the Network handles communication. The Chat client to the right will receive data through Bluetooth, since it was deemed best to service the data that way.

5.1 Client

The ChatClient class is a chat GUI for observing and receiving messages from other clients. On Windows, the chat client is a JFrame with a text field and a message area. For the applications on Windows and Raspberry PI, Bluetooth is done with the BlueCove Java library [9]. This gives a lightweight Java API for working with Bluetooth stack calls. From BlueCove, the ChatClient can detect available Bluetooth devices. Once all of the network information is added, the ChatClient will finalize its connection to the server. If the connection is successful, the server will request a name for the client. That name will be a unique identifier for when other clients are added. As long as the name is unique, the server will accept the information and allow a thread for communication. Now, the Network handles both reading and printing information on the Protocols available. The user now has both Wi-Fi and Bluetooth to send data over but will not interface with the complex information associated with those communication services. Instead, the user simply instantiates and provides connection information to the classes BluetoothProtocol and WifiProtocol. Then, those objects add the Network as an observer. The ChatClient's network now observes behaviors of the communication services without having the user worry about connectivity issues. All that is passed to the network will be strings of information displayed on the ChatClient's message area.

5.2 Server

For the server side of this application, a ChatServer class was created to seamlessly broadcast client information on either Bluetooth or Wi-Fi. The server is run only on Raspberry Pi or Windows as Android support was not necessary to test the application. Once run, the ChatServer first opens up a dialog for developers to choose what kind of connections should it accept. It can accept Wi-Fi and Bluetooth at the same time or each one individually. Once a client is accepted, a new thread will handle data sent to and from that client. Simply put, the Handler thread is a stripped down chat client. The constructor will get the socket information for the client and create a WifiProtocol and BluetoothProtocol object to manage it. The Network then observes the Protocols to keep them active, same as the ChatClient class. This makes the ChatServer simpler to understand and modify, not having to worry about the network information.

6 Related Work

Communication is a layer that all devices have to deal with but now, more than ever, the security of data from household devices and health systems makes it increasingly important [10]. This communication layer is one that has to work well with both the IoT cloud and the local embedded hardware systems. Cloud services take the work out of storing and manipulating data leaving the Software Developer to figure out how to both create IoT hardware and securely upload the data to the cloud. This framework, ReliableConnection, was created to bridge the gap between the hardware and the cloud and help developers focus on the logic of their system rather than the reliability of data flow. This frameworks also allows for new applications to develop since data can be extracted in new locations. If one protocol such as Wi-Fi is out of reach, possibly Bluetooth can find local devices to hop over in a sort of P2P style system. This system provides new possibilities as well as enhances IoT communication by offering a framework to make IoT a better, more connected reality.

7 Conclusion and Future Work

The Network and Protocol framework allow IoT applications a standardized way of sending data over a network, utilizing the advantages of multiple communication methods. This helps create applications that have increased trust in messages going through since a single Protocol failure will not halt communication.

For the future of this framework, more protocols such as RFID or IR could be added to allow new communication features or triggers. The Network could also be refined to monitor more details about each Protocol. Monitoring the speed and efficiency of each Protocol could assist in delivering information while reducing energy consumption and computational load.

References

1. Giusto, D., Lera, A., Morabito, G., Atzori, L. (eds.): The Internet of Things (2010)
2. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): a vision, architectural elements, and future directions. Future Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
3. World Health Organization: mHealth: New Horizon for Health Through Mobile Technologies (Global Observatory for e-health Services), vol. 3. WHO, Geneva (2011)
4. Niyato, D., Hossain, E., Diamond, J.: IEEE 802.16/WiMAX-based broadband wireless access and its application for telemedicine and e-health services. IEEE Wirel. Commun. Mag. **14**(1), 104–111 (2010)
5. Istepanaian, R.S., Zhang, Y.-T.: Guest editorial introduction to the special section: 4G health-the long-term evolution of m-health. IEEE Trans. Inf. Technol. Biomed. **16**(1), 1–5 (2012)
6. "FitBit." Fitbit Official Site for Activity Trackers & More. N.p., n.d. Web: 30 May 2016
7. "UP by Jawbone—Fitness Trackers for a Healthier You." Jawbone. N.p., n.d. Web: 30 May 2016
8. "Misfit." Misfit—Wearables, Activity Trackers, Fitness and Sleep Monitors. N.p., n.d. Web: 30 May 2016
9. Desmedt, Y.: Man-in-the-middle attack. In: Tilborg, H.C.A., Jajodia, S. (eds.) Encyclopedia of Cryptography and Security, pp. 759–759. Springer, US (2011)
10. Wortmann, F., Flüchter, K.: Internet of things. Bus. Inf. Syst. Eng. **57**(3), 221–224 (2015)